

# A Multipath TCP model for ns-3 simulator

Bachir Chihani  
Orange Labs  
Sophia Antipolis, France  
bachir.chihani@orange-ftgroup.com

Denis Collange  
Orange Labs  
Sophia Antipolis, France  
denis.collange@orange-ftgroup.com

## ABSTRACT

We present an implementation of Multipath TCP (MPTCP) under the NS-3 open source network simulator. MPTCP is a promising extension of TCP currently considered by the recent eponymous IETF working group, with the objective of improving the performance of TCP, especially its robustness to variable network conditions. We describe this new protocol, its main functions and our implementation in NS-3. Besides this implementation compliant to the current versions of the IETF drafts, we have also added and compared various packet reordering mechanisms. We indeed notice that such mechanisms highly improve the performance of MPTCP. We believe that our implementation could be useful for future works in MPTCP performance evaluation, especially to compare packet reordering algorithms or coupling congestion control mechanisms between subflows.

## Categories and Subject Descriptors

C.2.1 [Computer-Communication Networks]: Network Architecture and Design—*Multipath protocols*

## Keywords

Network simulation, ns-3, Multipath TCP, Packet Reordering

## 1. INTRODUCTION

Nowadays mobile equipment have often more than one single network interface. For instance, laptops have usually at least both a wired (Ethernet) and a wireless (Wifi) network adapters. Similarly smartphones and tablet PCs can reach the Internet either through Wifi or through a cellular network (UMTS or 3G+).

Another fact is that network operators usually duplicate links and equipments in order to protect their networks against failures, especially in the access and the backhaul networks. Moreover the backbone networks are generally meshed. In this context many paths may exist between any

two endpoints. The idea to use concurrently many paths has then emerged, to improve the robustness and performance of end-to-end connections. Such multipath connections can indeed balance the load between the different paths, switching dynamically and automatically the traffic from congested, disrupted or broken links to the best paths.

A lot of studies have considered the implementation of multipath capabilities at different layers: at the application layer [5], at the transport layer [13], [6], [1], [17], [20], [16], [6], etc. Following these last references, and [19], we think that the transport layer is a good place to implement multipath capabilities.

At this layer, end-systems can gather information about each used path: capacity, latency, congestion state. These information can then be used to react to congestion events in the network by moving the traffic away from congested paths. An IETF's working group has recently been created to specify a multipath protocol at the transport layer Multipath TCP [3] (MPTCP). More precisely, this working group develop protocol extensions to Transmission Control Protocol (TCP), the most used transport protocol on Internet, to handle multiple sub-connections following different paths between two endpoints.

Previous works in simulating MPTCP [7] focused on congestion control mechanisms without implementing other parts of MPTCP. This restriction was inherited from the used simulator, which does not allow the creation of an accurate model of the simulated protocol. Our contribution consists of the proposal of a model of MPTCP with a better fidelity, and also the proposal of enhancing the current version of MPTCP with packet reordering mechanisms to cope with the variety of path characteristics.

In this paper we first describe MPTCP in section 2, as it is specified in the current versions of the IETF drafts [3]. We describe in section 3 the implementation of two packet reordering algorithms, and how to adapt them to MPTCP. Then we present in section 4 our implementation of MPTCP under NS-3, conform to these IETF drafts, and with some of the congestion control algorithms proposed for MPTCP, for example in [14]. In section 5 we describe the simulation scenario, and comment the behaviour of the congestion window according to the used packet reordering algorithm. Finally, in section 6 we conclude the paper.

## 2. MULTIPATH TCP

An IETF working group has recently been created to specify a multipath protocol for the transport layer. They propose MPTCP [3] (Multipath TCP), an extension of TCP to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Wns3 2011 March 25, Barcelona, Spain.

Copyright 2011 ACM ...\$10.00.

handle multiple paths between two endpoints. MPTCP is designed with three major goals:

1. **Improve throughput:** the performance of a multi-path flow should be at least as good as this of a single-path flow on the best route.
2. **Do no harm:** a multi-path flow should not take up any more capacity on any one of its paths than a single-path flow using that route.
3. **Balance congestion:** a multi-path flow should move as much traffic as possible away from the most congested paths.

## 2.1 Main mechanisms

With MPTCP, the transport layer is splitted into two sub-layers. The upper one gathers the functionalities for connection management (establishing connection, reordering packets, etc.). The lower sublayer manages a set of subflows that can be seen each as one single TCP flow. MPTCP distinguishes two spaces of sequence numbers, one for each sublayer. Each subflow has its own sequence space which is similar to the Standard TCP sequence number, identifying bytes within a subflow. At the connection level, another sequence space is used to reorder the TCP segments before sending them to the application layer.

The MPTCP protocol uses new TCP options to exchange signalling information between peers: especially:

**MPC** (Multipath Capable) is used during the three-way handshake to establish a multipath TCP connection.

**DATA FIN** is used to inform the remote peer of the end of data and to close the multipath TCP connection.

**ADD** and **REMOVE** Address (IPv4) are used to inform the remote peer of the availability of a new address or to ask it to ignore an existing one.

**JOIN** is used to initiate a new sub-flow (packet flow on a route) between a not already used couple of addresses.

**DSN** (Data Sequence Number) is used as a map between the subflow level and the data sequence space number.

### 2.1.1 Connection establishment

Figure 1 illustrates the process of establishment of a MPTCP connection. After that the source application sends a Connect() call, the transport layer establishes a connection with the destination peer which was waiting for receiving connection requests. The establishment is TCP-like (three way handshake) with the use of MPC option to inform the callee that the initiator is able to exchange data using Multipath TCP. To initiate a new subflow, the peers must first exchange their additional IP addresses. The current MPTCP draft do not specify how the exchange may happen. We have chosen to send additional TCP segments. These segments handle the ADDR (Add address) option and they are sent just after the successful establishment of the MPTCP connection.

### 2.1.2 Subflow initiation

Figure 2 shows the initiation of a new subflow and the presence of a JOIN in a SYN segment. To maximize the chance that the subflow under initiation takes a path which is disjoint with previously established paths, each IP address is only used by one subflow.

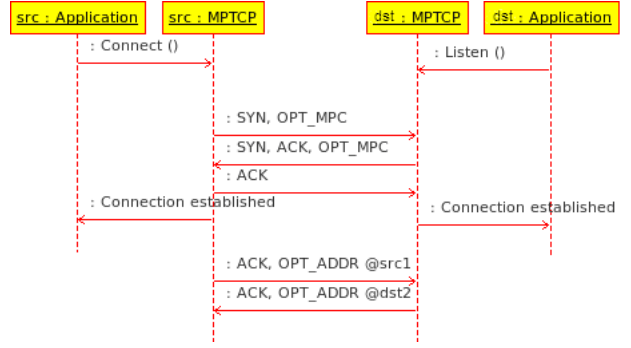


Figure 1: MPTCP connection establishment

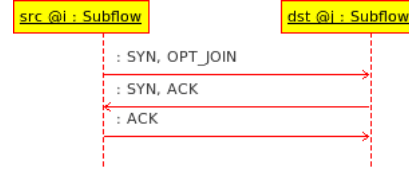


Figure 2: MPTCP sub-flow initiation

## 2.2 Traffic control

MPTCP redefines some TCP mechanisms so that they fit the multipath context. Congestion control allows the sender to regulate its throughput according to the available network resources. With MPTCP, the congestion control is performed at the subflow level. Each subflow has its own congestion window. However the congestion windows of the different subflows of a given TCP connection may be coupled to improve its performance. Besides, at the upper sublayer, the MPTCP receiver has a single global receiving window shared between the set of the established subflows. The objective is to do not limit the speed of some subflows. Four different algorithms have been proposed by Raiciu et al. in [14], coupling in various ways the congestion windows of active subflows: Uncoupled, Fully Coupled, Linked Increase, and RTT Compensator. They consider a simple extension of the standard congestion control mechanism TCP Reno in case the round-trip time is the same for all the available paths  $r = 1, \dots, N$ .

With the algorithm Uncoupled, the congestion window of each subflow behaves like for a single Standard TCP connection. Let  $w_r$  be the congestion window on path  $r$ , and  $w = \sum_r w_r$

**Algorithm Fully Coupled**

- $w_r = w_r + \frac{1}{w}$  per ACK on path  $r$
- $w_r = \max(w_r - \frac{w}{2}, 1)$  per loss event on path  $r$

Most of the time either one path or another is used with this algorithm, and rarely both. This phenomenon is called "flappiness". To reduce this flappiness, the authors proposed the following algorithm:

**Algorithm Linked Increases** [15]

- $w_r = w_r + \frac{a}{w}$  per ACK on path  $r$
- $w_r = \frac{w_r}{2}$  per loss event on path  $r$

In more general case when the round-trip times are not equal for the all paths, the authors adjust the precedent algorithm:

**Algorithm RTT Compensator**

- $w_r = w_r + \min(\frac{a}{w}, \frac{1}{w})$  per ACK on path  $r$
- $w_r = \frac{w_r}{2}$  per loss event on path  $r$

### 3. PACKET REORDERING

With Standard TCP, in networks with large packet jitter, i.e. when the end-to-end delay may vary a lot, packets may arrive out-of-sequence. This may for example be the case on wireless networks where mobile devices may change the used hotspot for accessing the Internet. The reordering of a packet makes the receiver responding with duplicated acknowledgements, and this may induce the sender to infer wrongly a packet loss. To avoid this problem and to distinguish clearly between packet losses due to congestion in the network and reordering due to transmission jitter, many mechanisms have been proposed for TCP. Leung et al. cite some of such mechanisms in [8]. Some of these mechanisms have been specified by the IETF [9] [2] [18], and some of them are implemented in the operating systems [18] (especially in the most frequent like Windows, Linux...) where they may be active by default, or not.

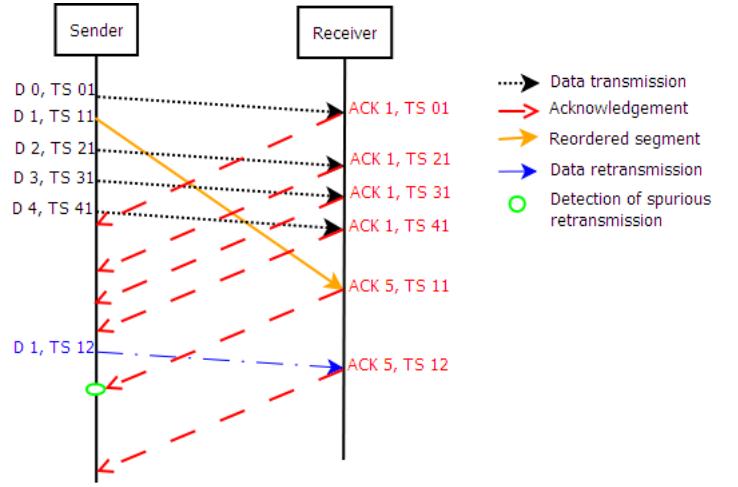
In multipath context, packets may also arrive out-of-sequence as the different paths may have different characteristics (especially the end-to-end delay), or congestion state (and then different queuing delays). The out-of-sequence arrival will create a problem for MPTCP while re-assembling packets at the connection level, and not at the subflow level because subflows are independent. So we suggest to consider for MPTCP similar reordering mechanisms than those proposed for Standard TCP on wireless networks.

Some of the algorithms, like Eifel [9] [10] and DSACK [2], need to store the connection state (the congestion window  $cwnd$ , the slow-start threshold  $ssthresh$ , etc.) before retransmitting. When a spurious retransmission is detected, the saved state is restored. Other algorithms, like the Blanton-Allman algorithm [8] and the RR-TCP (Reordering-robust TCP) [8], adjust the threshold  $dupthresh$  which is the number of received duplicated acknowledgement after which the sender considers a packet as lost and retransmits it. Others proposed mechanisms make the receiver delaying the transmission of duplicated ACK, like Paxson algorithm [8], or make the sender delaying its response to congestion (reception of three duplicated ACK), like TCP-DCR (Delayed Congestion Response TCP) [8].

We introduce in the following subsections some standardized algorithms [8] used by TCP to avoid packet reordering problems.

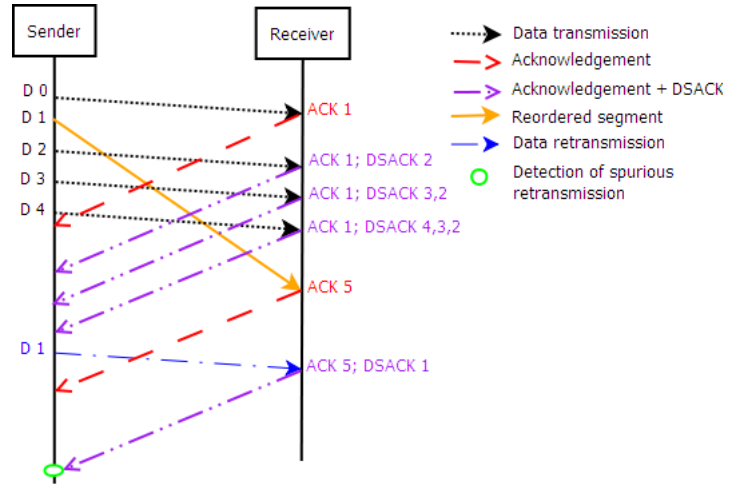
#### 3.1 Eifel Algorithm

Figure 3 illustrates how Eifel algorithm [9] [10] works. The sender inserts a TCP timestamp option in each transmitted segment, and the receiver inserts the timestamp value of the received segment in the corresponding acknowledgement. In case of loss, the sender saves the values of the current congestion window ( $cwnd$ ) and of the slow start threshold ( $ssthresh$ ). Then the sender retransmits the missing segment and stores its timestamp value. When the sender receives an acknowledgement for a retransmitted segment, it compares the saved timestamp value with the one inserted in



**Figure 3: Eifel Algorithm**

the acknowledgement. If the first one is greater then the retransmission is considered spurious and the values of  $cwnd$  and  $ssthresh$  are restored.



**Figure 4: DSACK Algorithm**

#### 3.2 DSACK Algorithm

Figure 4 illustrates how the DSACK algorithm [2] works. This algorithm is based on the SACK (Selective Acknowledgement) option. At the reception of a segment that creates a hole in the sequence numbers, the receiver sends back a duplicated acknowledgement containing a SACK option. The first block in the SACK option refers to the segment which triggers this duplicated acknowledgements. After three duplicated acknowledgements, the sender retransmits the missing segment, saves the congestion window value, and then enters a Congestion Avoidance phase. After that, when the sender detects that the retransmitted segment was acknowledged twice, it infers a spurious retransmission and begin a DSACK slow start to the stored congestion window's value.

## 4. IMPLEMENTATION

We have implemented MPTCP under the network simulator NS-3 [12] to evaluate MPTCP's performance and to compare the different congestion control mechanisms proposed in [14], and also to analyze the efficiency of our own mechanisms, such as those presented in the next section. The project is hosted at Google Code [11]. We opted for this simulator for its many interesting features. NS-3 is an open source network simulator, it is available for free under the GNU GPLv2 [4] license. NS-3 has a large community of developers and users, and it is the main network simulator used in the academic area.

In the following subsections, we illustrate the different states a MPTCP connection can take, the structure of our MPTCP implementation, the segment flow through the different used classes and the implemented algorithms for packet reordering.

### 4.1 Connection states

The MPTCP draft does not define for the moment any diagram to describe the different possible states of an MPTCP connection and their transitions. However, subflows may have the same states as a Standard TCP connection. For simplification, we have implemented a subset of the TCP states as described in figure 5. This figure illustrates the state diagram for a MPTCP subflow to exchange data, establish and close a connection. The starting and ending states are the state CLOSED.

The ESTABLISHED state allows data transfer between endpoints. Transitions to this state correspond to a connection opening, while transitions from it correspond to a connection closing on the corresponding subflow.

To send data, the application on the source host asks the transition from CLOSE state to SYN-SENT state – where the host has sent a SYN request and is waiting for a SYN-ACK answer. When the source host receives a SYN-ACK segment, the connection moves to ESTABLISHED state and an ACK segment is sent.

The connection on the destination host moves from CLOSE state to LISTEN state, after a passive opening by the application. When the destination receives a SYN segment, the connection moves from LISTEN to SYN-RCVD state and a SYN ACK segment is sent back to the source.

The application can request to close the connection. The connection is closed by moving from any state to CLOSING state and sending a FIN segment. After the receipt of a FIN ACK segment, the connection is totally closed and moved to CLOSED state.

### 4.2 Structure

The multipath transport layer is divided into two sublayers. The upper sublayer is responsible for the connection management: establishing connection, initiating subflows, etc. It is called the MPTCP sub-layer. The lower sub-layer is responsible for the management of each sub-flow.

The following classes (figure 6) are the main ones composing the MPTCP transport layer:

- **MpTcpSocketImpl** is a subclass of the NS-3 class **TcpSocketImpl**. It provides to the application layer a MPTCP API (connect, bind, etc.) to manage Multipath TCP connections. It also implements the packet reordering algorithms described previously.

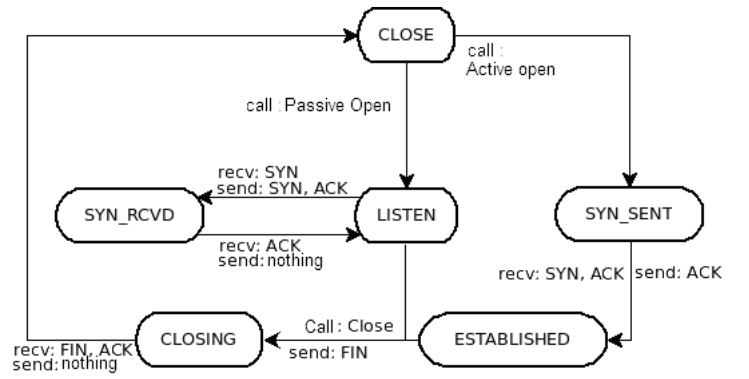


Figure 5: MPTCP connection's states

- **MpTcpL4Protocol** is a subclass of the NS-3 class **TcpL4Protocol**. It is an interface between the multipath transport layer and the network layer.
- **MpTcpSubflow** represents a subflow of a MPTCP connection.
- **MpTcpHeader** is a subclass of **TcpHeader**. An instance of this class is a TCP header that can handle options like TimeStamp, MPC, etc.

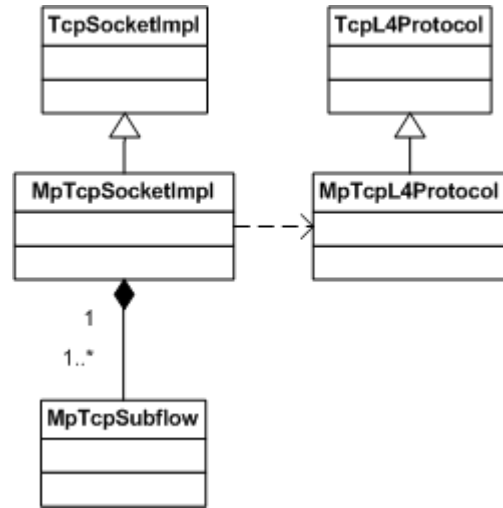


Figure 6: A subset of MPTCP classes

### 4.3 Segment flow

Figure 7 shows the structure of the multipath transport layer and its interfaces with the application and the network layers. The application and the multipath transport layers use the port number as an interface to communicate with each other. Similarly, the network and the multipath transport layers use the protocol number.

When the application has data to send, it chooses the appropriate transport layer instance using the peer source and destination port number. The **MpTcpSocketImpl** receives the data, splits it into TCP segments of a maximum size MSS<sup>1</sup> and then forwards it to the appropriate subflow which is

<sup>1</sup>MSS: Maximum Segment Size

represented by a `MpTcpSubflow` instance. At this level, a TCP segment header is created and will contain one of the MPTCP options (DSN option if the segment contains data). After that, the segment is ready to be sent, it is forwarded to the `MpTcpL4Protocol` which will forward it to the network layer.

At the receiver side, when the network layer receives a packet, it figures out the corresponding protocol number and sends the segment to the appropriate transport instance. When the `MpTcpL4Protocol` instance receives the segment, it uses the network layer addresses to find the corresponding subflow and then forward the segment to a `MpTcpSubflow` instance. The `MpTcpSubflow` updates the information related to its subflow (ex: sequence number), then it sends the segment to the `MpTcpSocketImpl`. The later adds the data to the previously received ones, notifies the application about the reception of new data and optionally generates a response (ex: sends back an acknowledgement to the source).

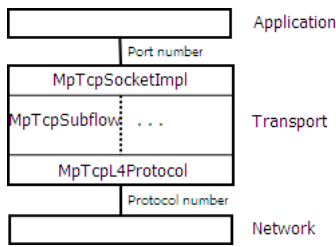


Figure 7: Multipath transport layer

## 4.4 Discussion

In NS-3, the transport layer is implemented via the `TcpSocketImpl` class. This class holds many variables like `m_endPoint` which is an instance of `Ipv4EndPoint`. This variable maintains information about a data flow (source - destination ports and addresses), and a callback for notification to higher layers that a packet from a lower layer was received.

When a packet is received by an instance of `TcpL4Protocol`, this one uses a callback (method `ForwardUp` of `m_endPoint`) to notify the upper layer (which is represented by `TcpSocketImpl`) of the reception. In fact, `TcpL4Protocol` holds a list of pointers to `Ipv4EndPoint` variables each one attached to a `TcpSocketImpl` instance.

In our MPTCP implementation, the classes composing the transport layer are derived from the ones of NS-3 (`TcpSocketImpl`, `TcpL4Protocol`).

The `MpTcpSocketImpl` class maintains a set of subflows, a received packet may belong to one of them and the `m_endPoint` variable is used to determine which one.

When `MpTcpL4Protocol` receives a packet, it updates the `m_endPoint` and notify the `MpTcpSocketImpl` about the reception of the packet.

The `MpTcpL4Protocol` may receive at the same time a lot of packets belonging to different subflows. In this case, we expected that the packets will be forwarded up one by one which make easy the use of the `m_endPoint` variable – it will contain the information of the right subflow. Instead of this, packets are forwarded up (after removing the IPv4 header) at once and not in the reception order. Thus, the `m_endPoint` variable will not contain a consistent information.

Such a behaviour is not a problem for Standard TCP. For

Multipath TCP it is because we can not decide to which subflow a packet belongs.

To overcome this, we used the sequence number in the segment header and for each subflow we gave a different sequence space.

## 5. SIMULATIONS

We used the implementation previously described to run a set of simulations in order to evaluate MPTCP connection performance by varying network parameters: bandwidth, latency and loss rate. Default values for these parameters are respectively: 0.5 Mb/s, 10 ms, 0. We also varied the used congestion control algorithm (Uncoupled TCPs, Linked Increases, etc.) and the packet reordering algorithm (none, Eifel, DSACK).

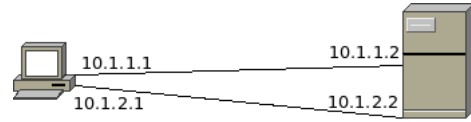


Figure 8: Simulated system

The simulated system (fig. 8) is composed of a FTP application, to transfer a 10 Go file, running on Client/Server architecture where the two hosts are linked by two point to point links.

Figure 9 shows the behaviour of the congestion window in case the Eifel algorithm is used. The graph of the congestion window of the subflow 1 oscillates between two curves of evolution: the congestion window takes its values according to the lower one in case of segment retransmission, and returns to the upper one when the Eifel algorithm detects that the retransmission was spurious.

Figure 10 illustrates the evolution of the congestion window of two subflows of a MPTCP connection in case the DSACK algorithm is used. In the graph of the congestion window of subflow 1, we can see three periods of time during which the corresponding window grows unusually in an exponential way. These periods reflect the DSACK Slow Start which is triggered by the DSACK algorithm after detecting a spurious retransmission.

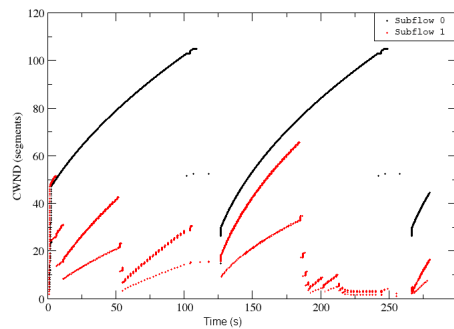
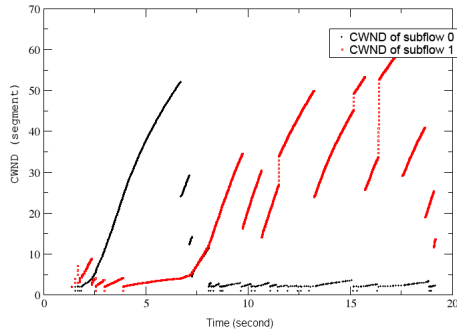


Figure 9: Congestion window evolution in case Eifel is used





**Figure 10: Congestion window evolution in case DSACK is used**

## 6. CONCLUSION

We have implemented the MPTCP protocol under the network simulator NS-3. The implementation is conform to the IETF drafts, at least according to their release in their July 2010. We also have added for this implementation of MPTCP two packet reordering mechanisms standardized for TCP.

We use now this implementation to test on simulations nonetheless under different realistic environments various congestion control and packet reordering mechanisms that could be used with the MPTCP protocol. Our objective to evaluate and compare the robustness and the performance of these different congestion control and packet reordering mechanisms.

Based on the conducted simulations, we deduced that neither Eifel nor DSACK will help to face effectively performance problems due to persistent out-of-sequence packets arrival. We plan to test other packet reordering mechanisms and assess their impact on avoiding or at least alleviating this problem.

## 7. REFERENCES

- [1] Y. Dong, N. Pissinou and J. Wang. "Concurrency Handling in TCP." 5th annual conference in Communication Networks and Services Research (CNSR'07), May 2007.
- [2] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky. "An Extension to the Selective Acknowledgement (SACK) Option for TCP," IETF RFC 2883. July 2000.
- [3] A. Ford, C. Raiciu and M. Handley, "TCP Extensions for Multipath Operation with Multiple Addresses," IETF draft draft-ietf-mptcp-multiaddressed-01, July 12, 2010.
- [4] GNU GPL (General Public License) version 2: <http://www.gnu.org/licenses/gpl-2.0.html>.
- [5] T. Hacker and B. Athey, "The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network," in IEEE Parallel and Distributed Processing Symposium., Proceedings International (IPDPS), Florida, April 2002.
- [6] Y. Hasegawa, I. Yamaguchi, T. Hama, H. Shimonishi and T. Murase. "Improved Data Distribution for Multipath TCP communication," IEEE Global

- Telecommunications Conference (Globecom), December 2005.
- [7] M. Handley and C. Raiciu, Multipath TCP implementations, [last accessed Jan. 23, 2011] <http://nrg.cs.ucl.ac.uk/mptcp/implementation.html>
- [8] Ka-Cheong Leung, Victor O.K. LI and D. Yang. "An Overview of Packet Reordering in Transmission Control Protocol (TCP) : Problems, Solutions, and Challenges". IEEE Transactions on Parallel and Distributed Systems, Vol. 18, No. 4, April 2007.
- [9] R. Ludwig, M. Meyer, "The Eifel Detection Algorithm for TCP," IETF RFC 3522, April 2003.
- [10] R. Ludwig, R. H.Katz. "The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions," ACM SIGCOMM Computer Comm. Review, vol. 30, no. 1, pp. 30-36, Jan 2000.
- [11] MPTCP Code Project: <http://code.google.com/p/mptcp-ns3/>
- [12] Network Simulator ns-3: <http://www.nsnam.org/>
- [13] L. Ong, J. Yoakum, "An Introduction to the Stream Control Transmission Protocol (SCTP)," IETF RFC 3286, May 2002.
- [14] C. Raiciu, D. Wischik and M. Handley, "Practical Congestion Control for Multipath Transport Protocols," UCL Technical Report, 2009.
- [15] C. Raiciu, M. Handley and D. Wischik, "Coupled Multipath-Aware Congestion Control", IETF draft draft-raiciu-mptcp-congestion-01, March 2010.
- [16] K. Rojviboonchai and H. Aida. "An evaluation of multi-path Transmission Control Protocol (M/TCP) with robust acknowledgement schemes," Internet Conference (IC'02), Tokyo (Japan), October 2002.
- [17] D. Sarkar. "A Concurrent Multipath TCP and Its Markov Model," IEEE International Conference on Communications (ICC), June 2006.
- [18] P. Sarolahti, M. Kojo, K. Yamamoto, M. Hata. "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP," IETF RFC 5682. September 2009.
- [19] D. Wischik, M. Handley and M. Bagnulo Braun, "The Resource Pooling Principle", ACM SIGCOMM Computer Communication Review, Vol 38.5, pp 47-52, October 2008.
- [20] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson and R. Wang. "A transport layer approach for improving end-to-end performance and robustness using redundant paths," Annual conference on USENIX (ATEC'04), Berkeley (CA, USA), June 2004.